НТЦ Метротек

Лекция №3. Языки описания аппаратуры. Часть 2

Толкачев Максим

m.tolkachev@metrotek.ru

НТЦ Метротек

План лекции

I. Что из SystemVerilog синтезируется

II. Основные синтезируемые конструкции SystemVerilog:
- always_ff
- always_comb
- Массивы (array)
- Конструкция if/else
- Тернарный оператор
- Конструкция case
- Параметры
- Пользовательские типы:
  - Структуры
  - Перечисляемый тип (enum)

III. Примеры модулей:
- Мультиплексор и демультиплексор
- Выделитель фронта
- Память
- Очередь (FIFO)
- Конечный автомат (FSM)

IV. Основные несинтезируемые конструкции SystemVerilog:
- Методы работы со временем
- Отображение информации
- Задачи и функции (task, function)
- Класс mailbox

НТЦ Метротек

I. Что из SystemVerilog синтезируется

НТЦ Метротек

Что можно использовать

Самые важные:

Конструкция	Описание
`always_ff @(posedge clk_i )`	Реализация регистров (см. **слайд № 6**)
`always_comb` или `assign`	Реализация комбинационной логики (см. **слайд № 8**)
Конструкция `if ... else`	Описание поведения при разных условиях (см. **слайд № 12**)
Тип `logic` и массивы (`logic [N:0] a;`)	См. **слайд № 15**
`module` и подключение модуля: `module_name inst_name ( <сигналы> );`	Модуль -- структурная единица всех проектов
Арифметические и логические операции	+,-,||,&& … (**см. лекцию № 2**)
Числа (`10`, `8'hff` ...)	Константы (см. **слайд № 19**)

Дополнительные:

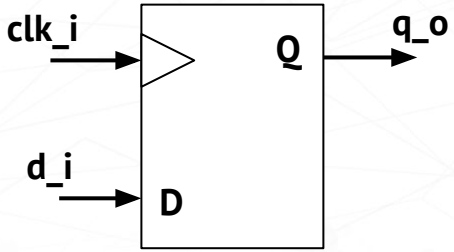
Конструкция	Описание
Конкатенация {}	См. **слайд № 20**
Тернарный оператор: (a) ? (b) : (c)	См. **слайд № 21**
Цикл `for`	См. **слайд № 23**
Конструкция `case`	См. **слайд № 25**
`parameter`	См. **слайд № 22**
`typedef`	См. **слайд № 26**
`struct`	См. **слайд № 27**
`enum`	См. **слайд № 28**
И много других!	

НТЦ Метротек

II. Основные синтезируемые конструкции SystemVerilog

always_ff

```
always_ff @(posedge clk_i)
  q_o <= d_i;
```




```
always_ff @(d_i)
  q_o <= d_i;
```

???

Синтаксически верно, но в FPGA реализовать невозможно!

Нет таких элементов!

НТЦ Метротек

always_ff

Базовые регистры. Описание D-триггера:

```
always_ff @(posedge clk_i )
  q_o <= d_i;
```

```
always_ff @(posedge clk_i or posedge rst_i )
  if( rst_i )
    q_o <= 0;
  else
    q_o <= d_i;
```

```
always_ff @(posedge clk_i or negedge rst_i )
  if( !rst_i )
    q_o <= 0;
  else
    q_o <= d_i;
```

Регистры + логика:

```
always_ff @(posedge clk_i )
  if( en )
    q_o <= d_i;
```

```
always_ff @(posedge clk_i )
  if( a == b )
    q_o <= 0;
  else
    q_o <= q_o + 1;
```

Несколько регистров в одном блоке:

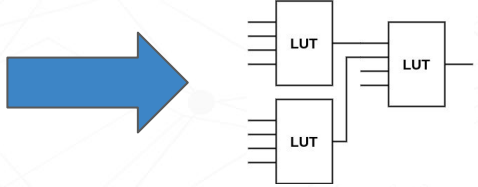
```
always_ff @(posedge clk_i )
  begin
    a <= b;
    b <= c;
  end
```

В перерывах между событиями сигналы сохраняют значения!

НТЦ Метротек

always_comb

```
always_comb
  a = b ^ c;
```



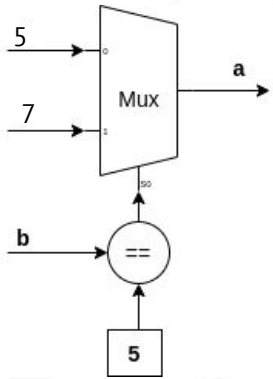

```
always_comb
  begin
    a = 5;
    a = 6;
    a = 7;
    a = 8;
  end
```

“Вычисляется” сверху вниз **постоянно**. Какое значение у сигналов к концу прохождения, такое оно и будет.

```
a = 8;
```

```
always_comb
  begin
    a = 5;
    if( b == 5 )
      a = 7;
  end
```

```
always_comb
  begin
    if( b == 5 )
      a = 7;
    else
      a = 5;
  end
```

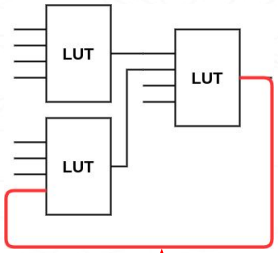


НТЦ Метротек

Что нельзя в always_comb

```
always_comb
  begin
    if( b == 5 )
      a = 7;
  end
```

Не описано, что делать, когда b != 5



Комбинационная петля

Так делать нельзя!

По правилам SystemVerilog получается, что если **else** нет, то значение должно остаться старым.
Но это комбинационная схема!

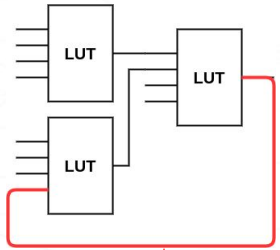
НТЦ Метротек

Что нельзя в always_comb

```
always_comb
  a = a + 7;
```

Использование сигнала для вычисления себя же.

```
always_comb
  b = a + 3;

always_comb
  a = b + 7;
```



Комбинационная петля

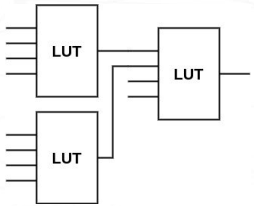
Так делать нельзя!

НТЦ Метротек

assign

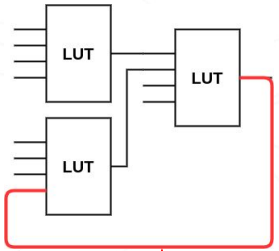
Как `always_comb` только для одного сигнала и в одну строку.

```
assign a = b ^ c;
```



Так делать можно

```
assign a = a + c;
```



Так делать нельзя!

Комбинационная петля

НТЦ Метротек

Конструкция if … else

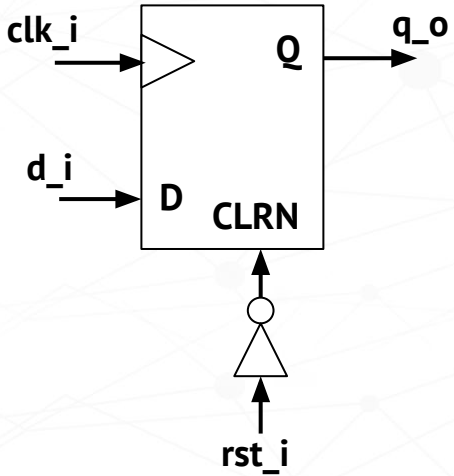
Мы описываем поведение, которое хотим. Синтезатор решает, как это реализовать.

Можно описывать как угодно, главное чтобы не противоречило правилам `always_ff` и `always_comb`

Мы хотим:

```
always_ff @(posedge clk_i or posedge rst_i )
  if( rst_i )
    q_o <= 0;
  else
    q_o <= d_i;
```

Синтезатор решил:



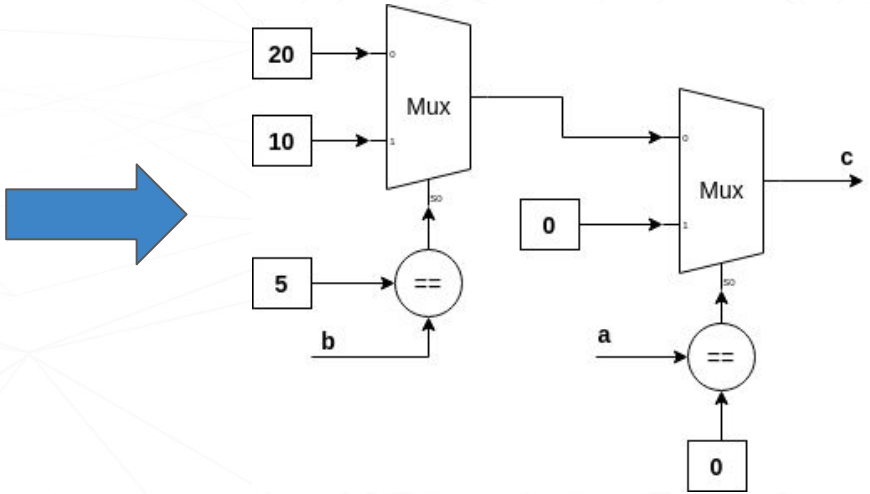
НТЦ Метротек

Конструкция if ... else

Мы хотим:

```
always_comb
  begin
    if( a == 0 )
      c = 0;
    else
      begin
        if( b == 5 )
          c = 10;
        else
          c = 20;
      end
  end
```

Синтезатор решил:



НТЦ Метротек

Конструкция if ... else

Условие True или False:

- Число != 0 -> **True**
- Число == 0 -> **False**
- logic уровень “1” -> **True**
- logic уровень “0”, “x”, “z” -> **False**

```
if( <условие> )
  begin
    <выражение №1>;
    <выражение №2>;
    ...
  end
else
  <выражение №3>;
```

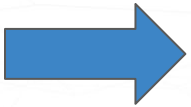
Если строк под условием больше одной -- нужны **begin** и **end**.

Часть **else** не обязательная

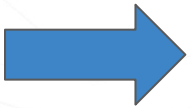
НТЦ Метротек

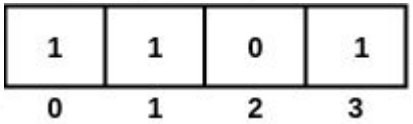
Массивы (array)

```
logic [3:0] a;
```




```
logic [3:0] a;

always_comb
  a = 4'b1011;
```



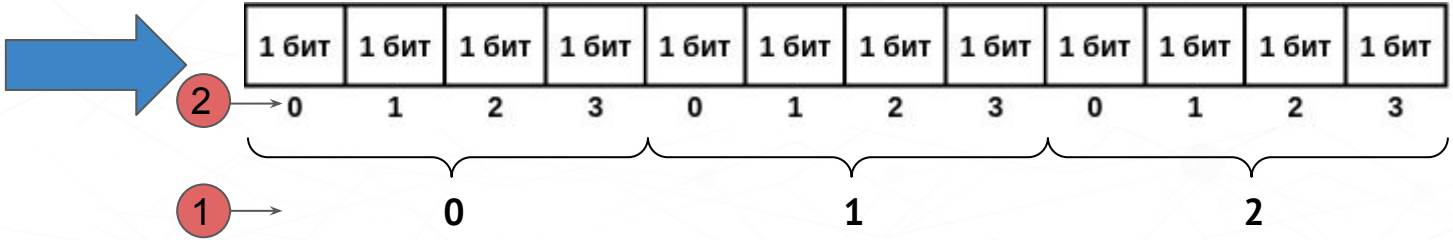


НТЦ Метротек

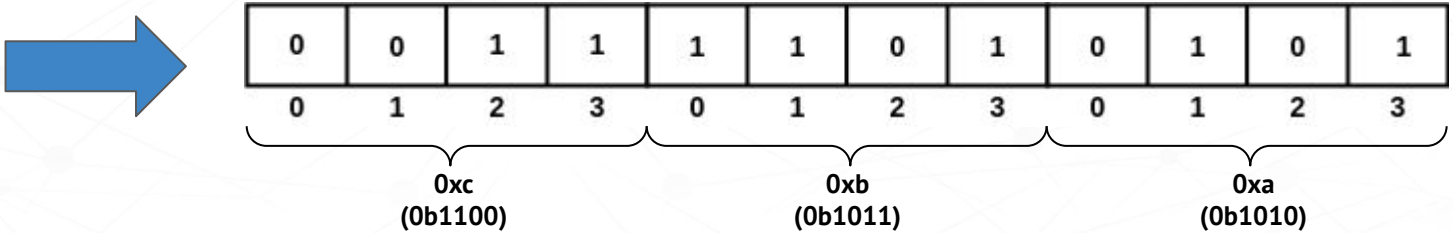
Массивы (array)

Двумерный массив:

```
logic [2:0][3:0] a;
```




```
logic [2:0][3:0] a;
assign a = 12'habc;
```



НТЦ Метротек

Массивы (array)

Packed (упакованные)

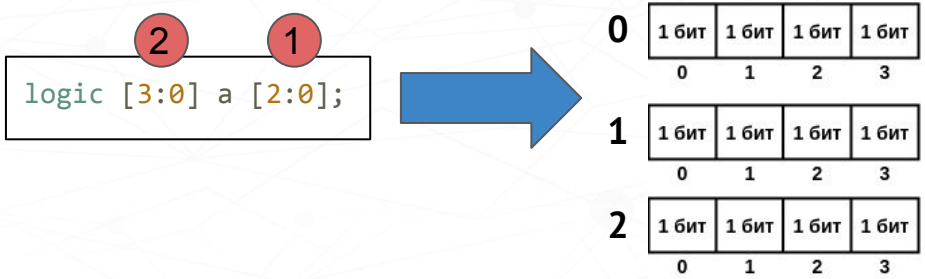
```
logic [2:0][3:0] a;
```

- Данные идут подряд:

```
logic [2:0][3:0] a;
logic [11:0] b;

assign b = a;
```

- Только синтезируемые типы:
  - `int [2:0][3:0] a;`
  - `bit [2:0][3:0] a;`
  - `integer [2:0][3:0] a;`
- Размер константный и конечный.

Unpacked (неупакованные)



Данные нельзя интерпретировать как одно число!

```
assign a = 12'habc;
```

- Любых типов (синтезируются только синтезируемые);
- Размер может быть бесконечным:

```
bit [7:0] a [$]; // называется очередь
```

НТЦ Метротек

Массивы (array): выбор части

```
logic [2:0][3:0] a;
```

- Выбрать 1 элемент:

```
logic [3:0] b;
assign b = a[0];

logic b;
assign b = a[1][3];
```

- Выбрать несколько элементов:

```
logic [1:0] b;
assign b = a[2][3:2];

logic b;
assign b = a[1][3];
```

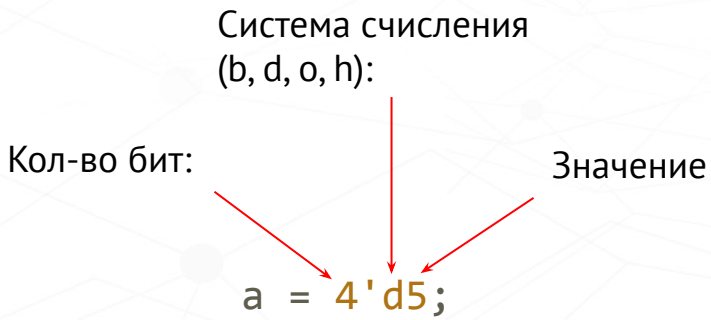
- Выбрать 1 элемент динамически:

```
logic [3:0] b;
logic [1:0] select; // 0, 1, 2, 3

assign b = a[ select ];
```

Выбрать несколько элементов динамически нельзя: размер выборки должен быть постоянным.

НТЦ Метротек

Числа (константы)



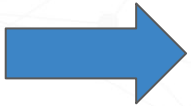

```
a = 5'b011011;

a = 10; // Знаковый int (32 бита)

a = 8'hff;

a = '0; // Все биты “0”

a = '1; // Все биты “1”

a = (WIDTH)'(1); // Из 32 бит младшие WIDTH
бит.
```

НТЦ Метротек

Конкатенация {<a>,<b>,...}

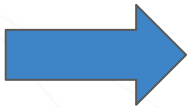
```
logic [1:0] b;
assign b = {1'b1, 1'b0};
```

Младший бит



```
b = 2;
```

```
logic c, d;
assign {c, d} = 2'b10;
```



```
c = 1;

d = 0;
```

НТЦ Метротек

Тернарный оператор

(<УСЛОВИЕ>) ? (<ЕСЛИ_ИСТИНА) : (<ЕСЛИ_ЛОЖЬ>)

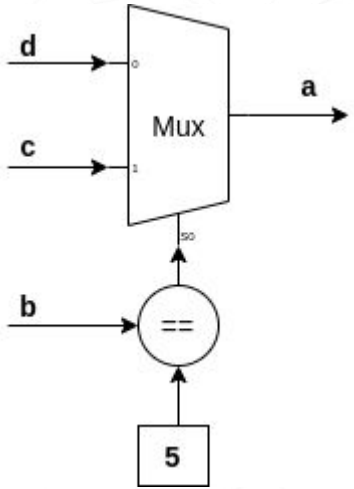
```
always_comb
  begin
    a = ( b == 5 ) ? ( c ) : ( d );
  end
```

или

```
assign a = ( b == 5 ) ? ( c ) : ( d );
```

==

```
always_comb
  begin
    if( b == 5 )
      a = c;
    else
      a = d;
  end
```



НТЦ Метротек

Параметры (parameter)

Параметры определяются во время синтеза. Не могу меняться динамически.

- `parameter` -- можно переопределить при инстансе
- `localparam` -- нельзя переопределить при инстансе

```
module my_module #(
  parameter I_WIDTH = 5,
  parameter O_WIDTH = 6
)(
  input  logic [I_WIDTH-1:0] data_i,
  output logic [O_WIDTH-1:0] data_o
);

parameter  CNT_WIDTH  = 7;

localparam WORD_WIDTH = 8;

...

endmodule
```

```
module my_top_module (
  input  logic [4:0] data_i,
  output logic [5:0] data_o
);

my_module #(
  .I_WIDTH   ( 5      ),
  .O_WIDTH   ( 6      ),
  .CNT_WIDTH ( 6      )
)(
  .data_i  ( data_i ),
  .data_o  ( data_o )
);

endmodule
```

Инстанс модуля

НТЦ Метротек

Цикл for

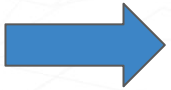
Аппаратно реализуются только циклы с **фиксированным числом итераций.**

Никаких счетчиков не будет:

`i` -- не станет счетчиком!

Цикл только для сокращения кода. Его всегда можно переписать (развернуть).

```
parameter A = 4;

logic [A-1:0][B-1:0] data;

always_comb
  begin
    for( int i = 0; i < A; i++)
      begin
        if( i < 2 )
          begin
              data[i] = (B)'(i);
            eles
              data[i] = (B)'(i-2);
          end
      end
  end
```



```
parameter A = 4;

logic [A-1:0][B-1:0] data;

always_comb
  begin
    data[0] = (B)'(0);
    data[1] = (B)'(1);
    data[2] = (B)'(0);
    data[3] = (B)'(1);
  end
```

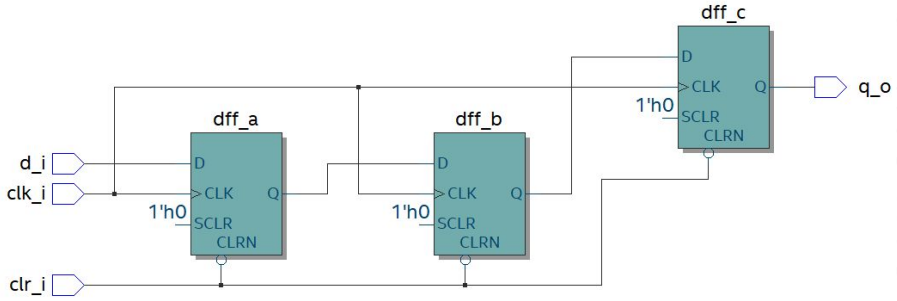
НТЦ Метротек

Цикл for

```
module my_module #(
  parameter CNT_W = 4
)(
  input  logic data_i,
  output logic data_o
);

logic [CNT_W-1:0] cnt;

always_ff @(posedge clk_i )
  begin
    cnt[0] <= data_i;
    for( int i = 1; i < CNT_W; i++)
      begin
        cnt[i] <= cnt[i-1];
      end
  end

endmodule
```

```
always_ff @(posedge clk_i )
  begin
    cnt[0] <= data_i;
    cnt[1] <= cnt[0];
    cnt[2] <= cnt[1];
    cnt[3] <= cnt[2];
  end
```

Сдвиговый регистр!



НТЦ Метротек

Конструкция case

```
case( <сигнал> )
  <Значение 1>:
    begin
      <выражение №1>;
      <выражение №2>;
    end
  <Значение 2>:
      <выражение №3>;
  ...
default:
      <выражение №4>;
endcase
```

==

```
if( <сигнал> == <Значение 1> )
  begin
    <выражение №1>;
    <выражение №2>;
  end
else
  begin
    if( <сигнал> == <Значение 2> )
      <выражение №3>;
    else
      <выражение №4>;
  end
```

```
always_comb
    case( a )
      1:
        b = 5;
      2:
        b = 8;
    default:
        b = 0;
    endcase
```

НТЦ Метротек

Пользовательские типы

`typedef` : можно собрать из стандартных типов свой тип:

```
typedef logic [2:0] logic_3_bits_t;

module my_module (
  input  logic [2:0] a,
  output logic       b
);

logic_3_bits_t c;

assign c = a;

...

endmodule
```

Создавать можно вне модуля или внутри модуля (но не внутри процедурных блоков);

== `logic [2:0] c;`

НТЦ Метротек

Структуры (structure)

`packed` или ничего (тогда `unpacked`)

`packed` -- синтезируются и поля друг за другом гарантированно.

```
typedef struct packed {
  logic [2:0] field_2; // старшие биты
  logic [2:0] field_1;
  logic [1:0] field_0; // младшие биты
} my_new_type_t;

module my_module (
  input  logic [7:0] data_i,
  output logic [2:0] data_o
);

my_new_type_t a;

assign a = data_i;
assign data_o = a.field_2;

endmodule
```

Создаем новый тип -- структуру.

```
typedef struct packed {
  logic [2:0] field_2;
  logic [2:0] field_1;
  logic [1:0] field_0;
} my_new_type_t;

module my_module (
  input  logic [$bits(my_new_type_t)-1:0] data_i,
);
...
```

Полезная “фича”

НТЦ Метротек

Перечисляемый тип (enum)

Базовый тип.
По умолчанию `int`.

```
typedef enum logic [1:0] {
  A,
  B = 2'b00,
  C
} my_enum_type_t;

module my_module (
  input  logic       clk_i,
  input  logic [7:0] data_i,
  output logic [2:0] data_o
);

my_enum_type_t result;

                  ...


endmodule
```

Создаем новый тип.

Можно задавать значения.

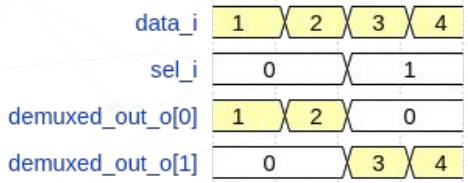
```
always_ff @( posedge clk_i )
  if( data_i == 1 )
    result <= A;
  else
    begin
      if( data_i == 2 )
        result <= B;
      else
        result <= C;
    end

assign data_o = ( result == B ) ? ( 2 ) : ( 0 );
```

НТЦ Метротек

III. Примеры модулей

НТЦ Метротек

Мультиплексор и демультиплексор




```
module mux #(
  parameter IN_CNT = 2,
  parameter DATA_W = 7
)(
  input  logic [IN_CNT-1:0][DATA_W-1:0] data_i,
  input  logic [$clog2(IN_CNT)-1:0]     sel_i,
  output logic [DATA_W-1:0]             muxed_out_o
);

assign muxed_out_o = data_i[ sel_i ];

endmodule
```

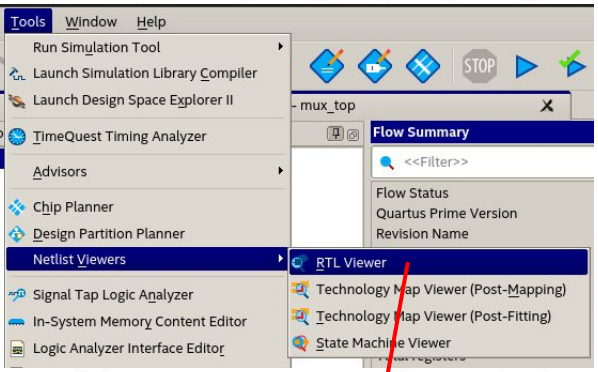
```
module demux #(
  parameter OUT_CNT = 2,
  parameter DATA_W  = 7
)(
  input  logic [DATA_W-1:0]              data_i,
  input  logic [$clog2(OUT_CNT)-1:0]     sel_i,
  output logic [OUT_CNT-1:0][DATA_W-1:0] demuxed_out_o
);
always_comb
  begin
    demuxed_out_o = '0;
    demuxed_out_o[ sel_i ] = data_i;
  end
endmodule
```

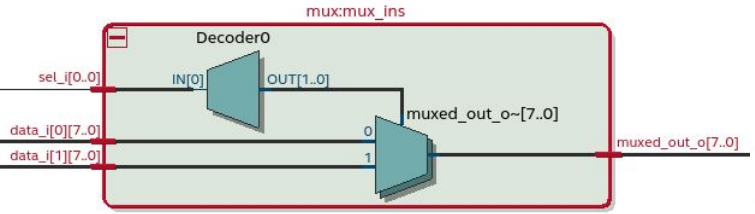
Проект на github: https://github.com/stcmtk/fpga-webinar-2020/tree/master/lecture_3/mux_demux

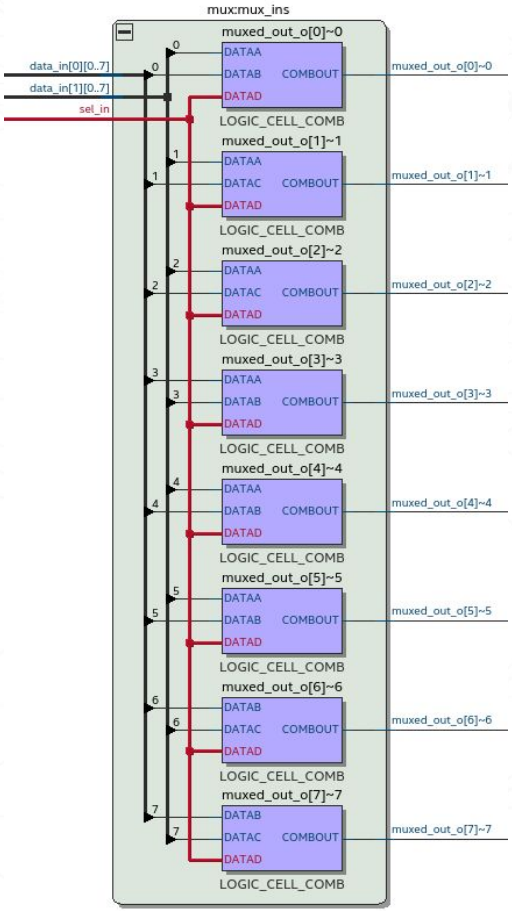
НТЦ Метротек

```
parameter IN_CNT = 2
parameter DATA_W = 8
```

Мультиплексор







НТЦ Метротек

Демультиплексор

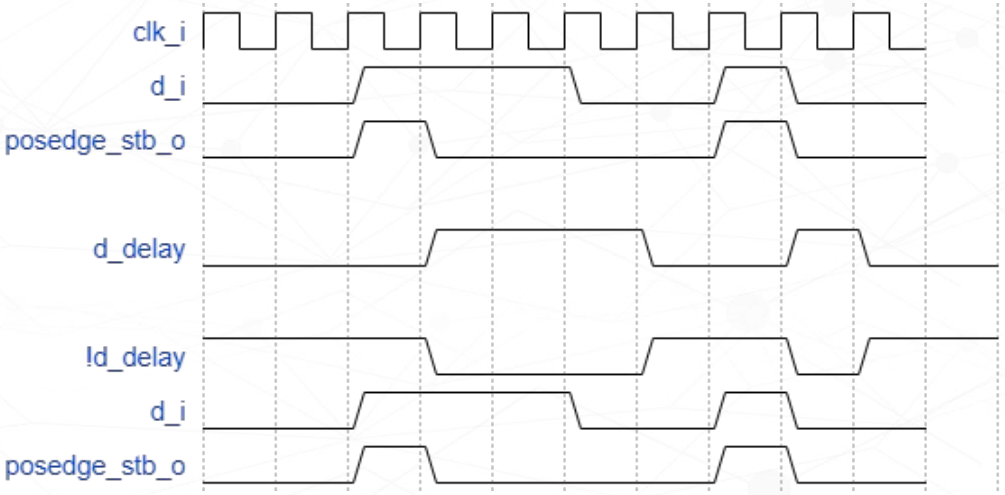
Домашняя работа: разобраться, как синтезируется демультиплексор.

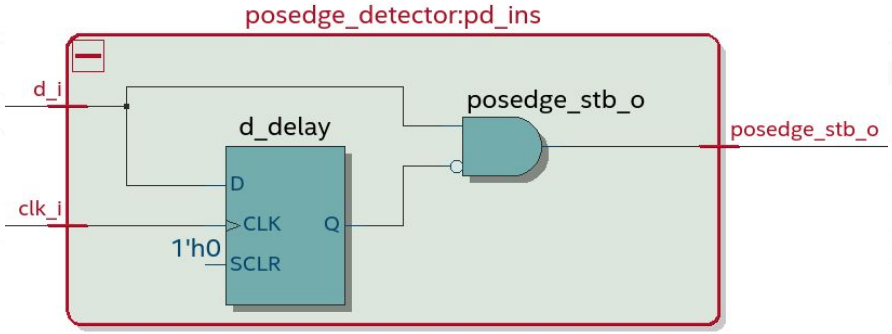
Проект на github:
https://github.com/stcmtk/fpga-webinar-2020/tree/master/lecture_3/mux_demux/demux_example

НТЦ Метротек

Выделитель фронта

```
module posedge_detector (
  input  logic clk_i,
  input  logic d_i,
  output logic posedge_stb_o
);

logic d_delay;

always_ff @( posedge clk_i )
  d_delay <= d_i;

assign posedge_stb_o = d_i && !d_delay;

endmodule
```

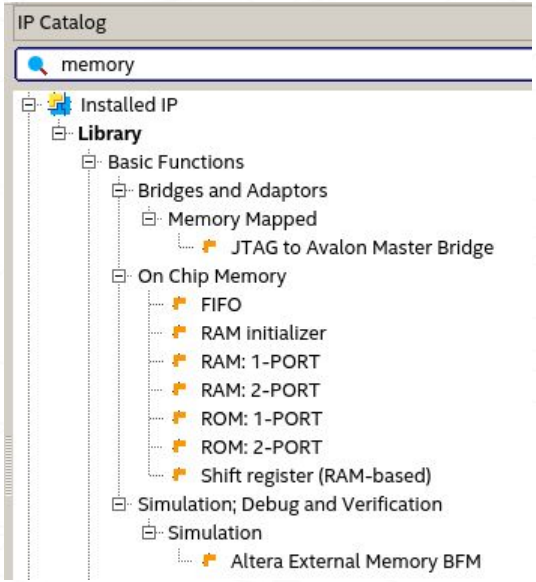




Проект на github: https://github.com/stcmtk/fpga-webinar-2020/tree/master/lecture_3/posedge_detector

Память

IP-core или template



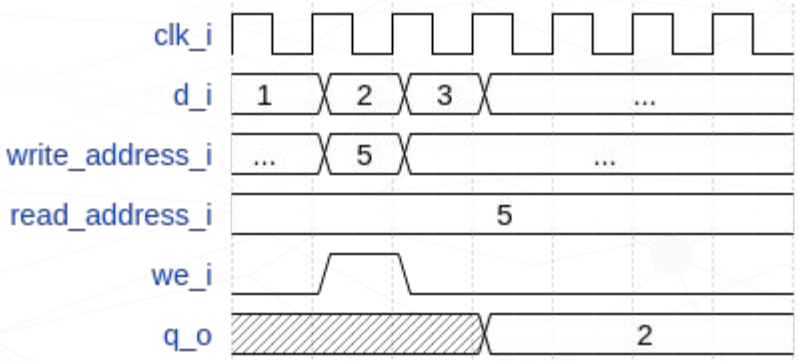
Inferred RAM

Intel Quartus Prime Pro Edition User Guide: Design Recommendations :
1.4. Inferring Memory Functions from HDL Code

Проект на github: https://github.com/stcmtk/fpga-webinar-2020/tree/master/lecture_3/memory/

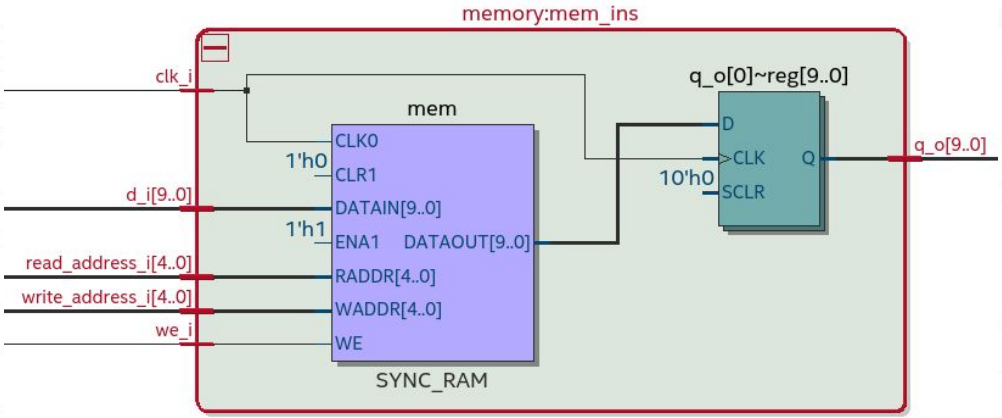
Inferred RAM

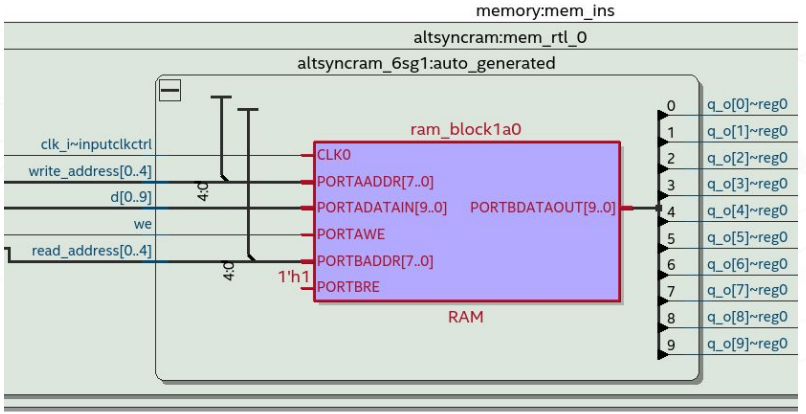
```
module memory #(
  parameter ADDR_W = 5,
  parameter DATA_W = 10
)(
  input logic                 clk_i,
  input logic [DATA_W-1:0]  d_i,
  input logic [ADDR_W-1:0]  write_address_i,
  input logic [ADDR_W-1:0]  read_address_i,
  input logic                 we_i,

  output logic [DATA_W-1:0] q_o
);

logic [DATA_W-1:0] mem [2**ADDR_W-1:0];

always_ff @( posedge clk_i )
  begin

    if( we_i )
      mem[write_address_i] <= d_i;

    q_o <= mem[read_address_i];
 end

endmodule
```



Inferred RAM

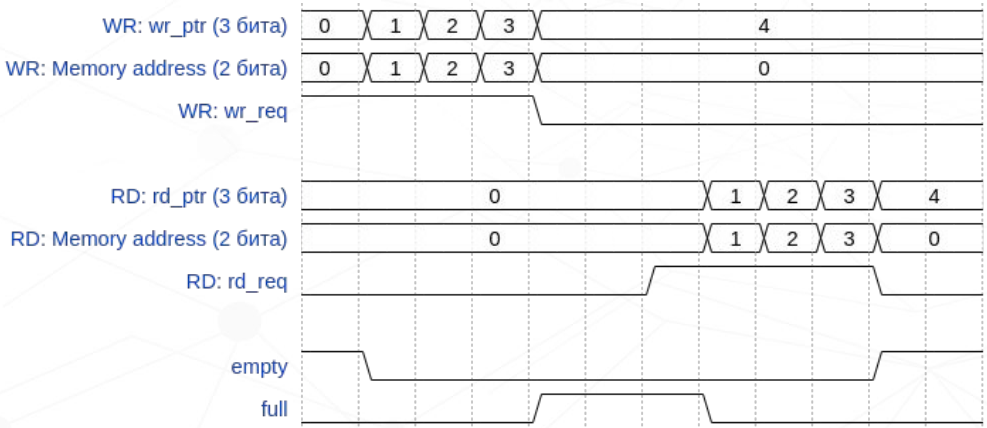
```
module memory #(
  parameter ADDR_W = 5,
  parameter DATA_W = 10
)(
  input logic                 clk_i,
  input logic [DATA_W-1:0]    d_i,
  input logic [ADDR_W-1:0]    write_address_i,
  input logic [ADDR_W-1:0]    read_address_i,
  input logic                 we_i,

  output logic [DATA_W-1:0] q_o
);

logic [DATA_W-1:0] mem [2**ADDR_W-1:0];

always_ff @( posedge clk_i )
  begin

    if( we_i )
      mem[write_address_i] <= d_i;

    q_o <= mem[read_address_i];
 end

endmodule
```

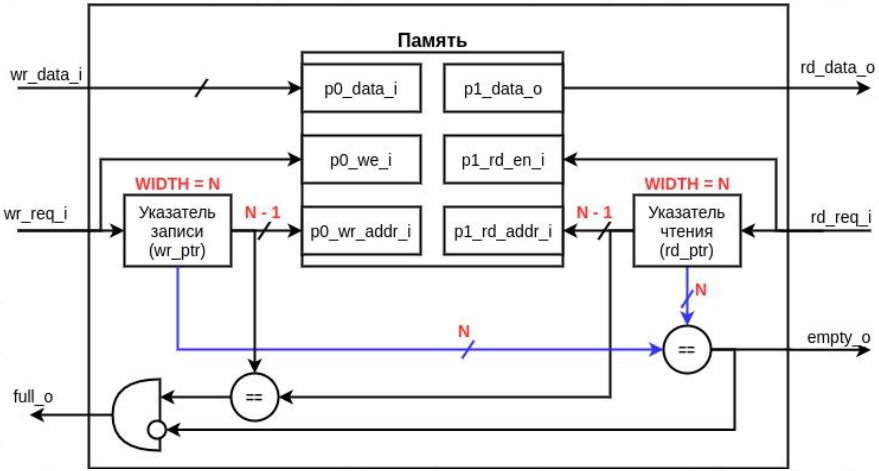




НТЦ Метротек

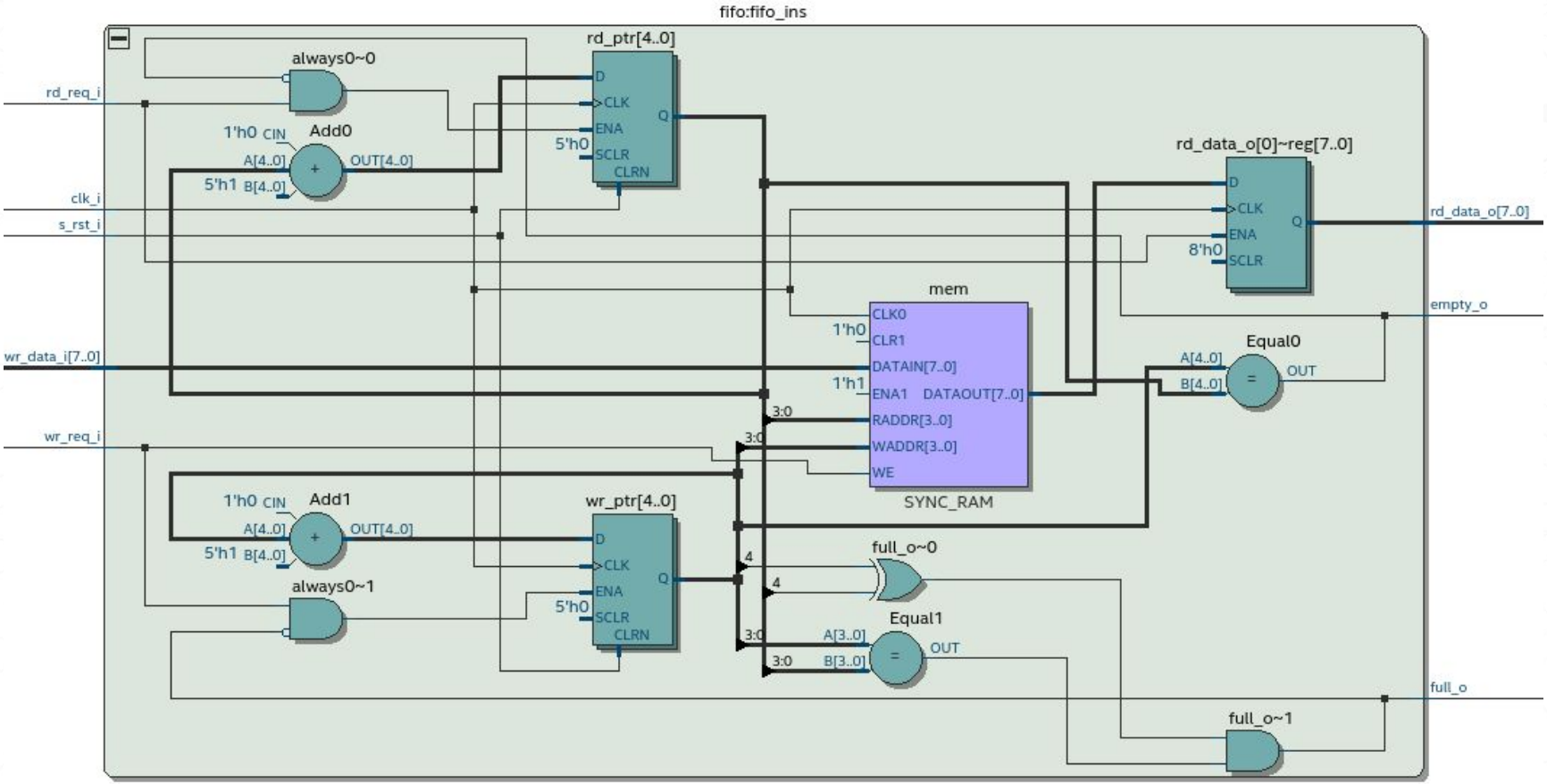
Очередь (FIFO)





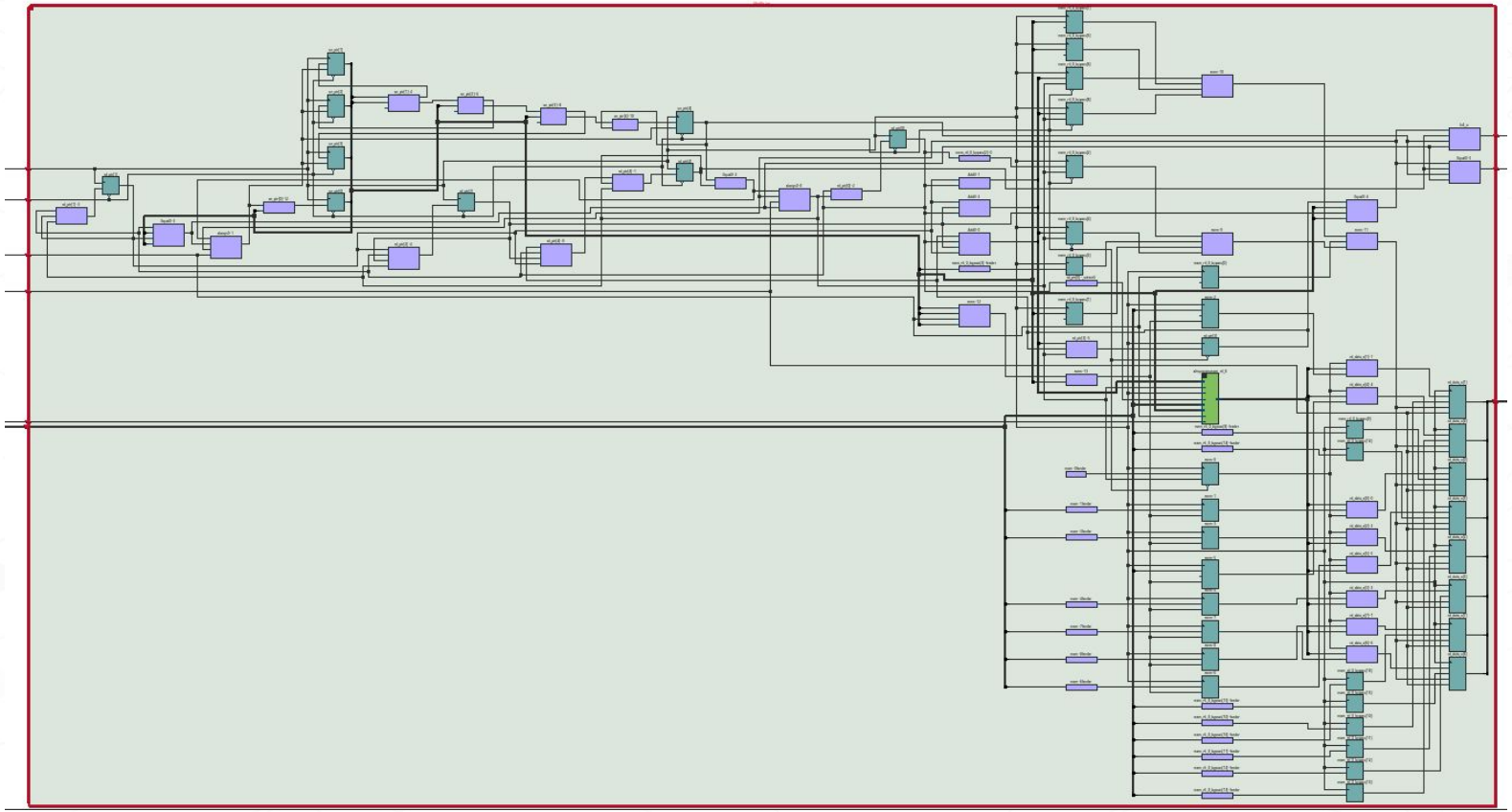
Проект на github: https://github.com/stcmtk/fpga-webinar-2020/tree/master/lecture_3/fifo

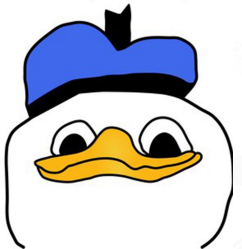
Очередь (FIFO)



НТЦ Метротек

Очередь (FIFO)





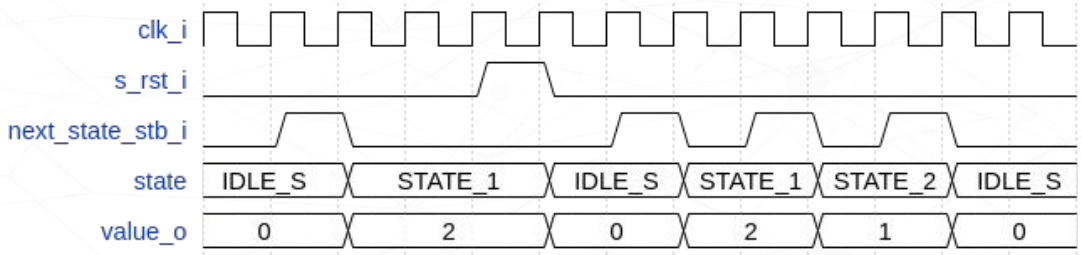
НТЦ Метротек

Конечный автомат (FSM)

```
module fsm(
  input logic         clk_i,
  input logic         s_rst_i,

  input  logic        next_state_stb_i,
  output logic [1:0] value_o
);
```

```
enum logic [1:0] {
  IDLE_S,
  STATE_1,
  STATE_2
} state, next_state;
```

```
always_ff @( posedge clk_i )
  if( s_rst_i )
    state <= IDLE_S;
  else
    state <= next_state;
```



1

Из **трех** `always` блоков

Проект на github: https://github.com/stcmtk/fpga-webinar-2020/tree/master/lecture_3/fsm

НТЦ Метротек

Конечный автомат (FSM)

2

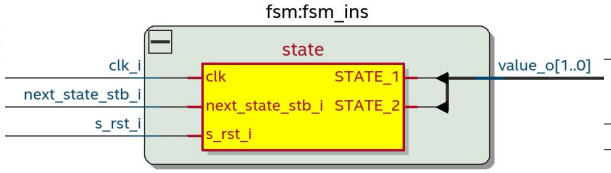
```
always_comb
  begin
    next_state = state;
    case( state )
      IDLE_S:
        begin
          if( next_state_stb_i )
            next_state = STATE_1;
        end
      STATE_1:
        begin
          if( next_state_stb_i )
            next_state = STATE_2;
        end
      STATE_2:
        begin
          if( next_state_stb_i )
            next_state = IDLE_S;
        end
    default:
        begin
          next_state = IDLE_S;
        end
    endcase
  end
```

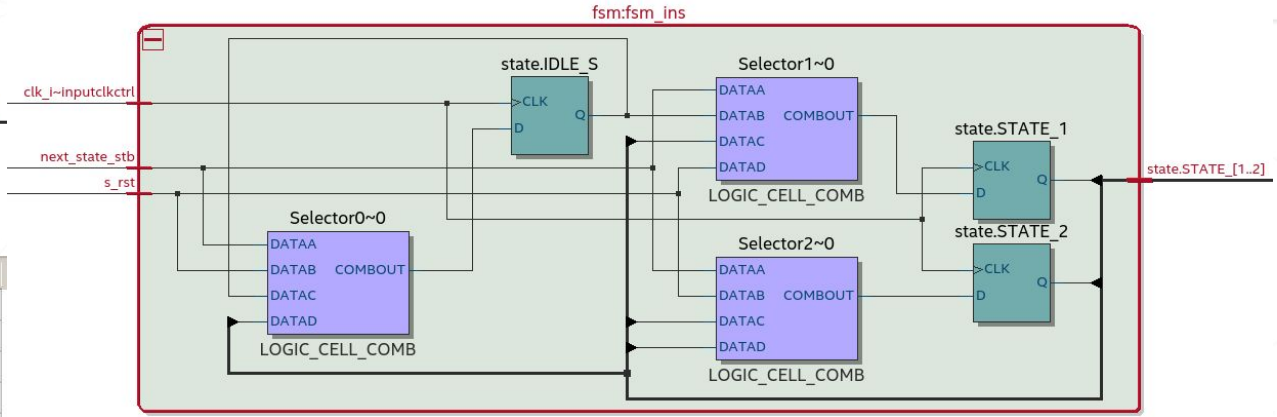
3

```
always_comb
  begin
    value_o = '0;
    case( state )
      IDLE_S:
        begin
          value_o = 2'd0;
        end
      STATE_1:
        begin
          value_o = 2'd2;
        end
      STATE_2:
        begin
          value_o = 2'd1;
        end
    default:
        begin
          value_o = 2'd0;
        end
    endcase
  end
```

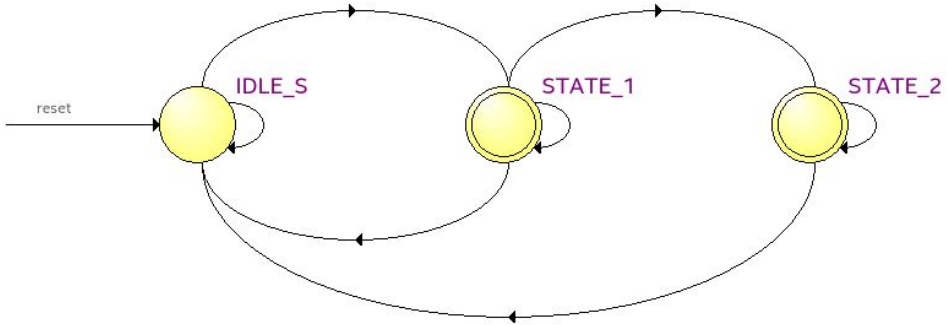
НТЦ Метротек

Конечный автомат (FSM)





	Source State	Destination State	Condition
1	IDLE_S	IDLE_S	(!next_state_stb_i) + (next_state_stb_i).(s_rst_i)
2	IDLE_S	STATE_1	(next_state_stb_i).(!s_rst_i)
3	STATE_1	IDLE_S	(s_rst_i)
4	STATE_1	STATE_1	(!next_state_stb_i).(!s_rst_i)
5	STATE_1	STATE_2	(next_state_stb_i).(!s_rst_i)
6	STATE_2	IDLE_S	(!next_state_stb_i).(s_rst_i) + (next_state_stb_i)
7	STATE_2	STATE_2	(!next_state_stb_i).(!s_rst_i)



Правильно описанный конечный автомат распознается в Quartus.

НТЦ Метротек

IV. Основные несинтезируемые конструкции SystemVerilog

НТЦ Метротек

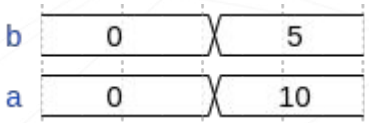
Методы работы со временем

-- задержка по умолчанию в нс

```
initial
  begin
    a = 0;
    #10;
    a = 5;
    #10;
    a = 10;
  end
```

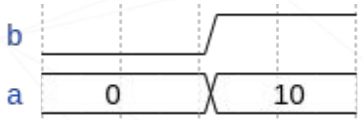
wait(<условие>)
Дождаться, когда условие станет “True”

```
initial
  begin
    a = 0;
    wait( b == 5 );
    a = 10;
  end
```



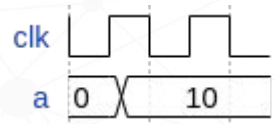
@(posedge <сигнал>)

```
initial
  begin
    a = 0;
    @( posedge b )
    a = 10;
  end
```



clocking block

```
bit   clk;

initial
  forever
    #5 clk = !clk;

default clocking cb
  @(posedge clk);
endclocking

initial
  begin
    a = 0;
    ##1;
    a = 10;
  end
```



НТЦ Метротек

Отображение информации

$monitor

```
initial
  begin
    a = 0;
    #10;
    a = 10;
    #12;
    a = 15;
  end

initial
  begin
    $monitor( "Value of a: %d in %d ns" , a, $time );
  end
```

Вывод transcript:

```
# Value of a:     0 in          0 ns
# Value of a:    10 in         10 ns
# Value of a:    15 in         22 ns
```

$display

```
initial
  begin
    a = 0;
    #10;
    a = 10;
    #12;
    a = 15;
  end

initial
  begin
    $display( "Value of a: %d in %d ns" , a, $time );
  end
```

Вывод transcript:

```
# Value of a:     0 in          0 ns
```

НТЦ Метротек

Задачи и функции (task, function)

task

```
task my_task (
  input  int a,
  input  int b,
  output int c
);
  wait( a == 5 );
  #10;
  c = a + b;
endtask
```

- **task** может содержать методы работы со временем.
- **task** не возвращает значений (но **return** можно использовать для завершения).

function

```
function int my_function (
  int a,
  int b
);

  return a + b;

endfunction
```

- **function** не может содержать методы работы со временем.
- **return** возвращает значение заданного при объявлении функции типа.
- **Функции можно использовать в синтезируемых блоках.**

НТЦ Метротек

Очередь ([$])

Очередь -- неупакованный бесконечный массив.

```
bit [7:0] a [$];
bit [7:0] b;

initial
  begin
    a.push_front(1); // a = [1]
    a.push_front(2); // a = [1,2]
    a.push_front(3); // a = [1,2,3]
    b = a[0];        // b = 3
    b = a[$];        // b = 1
    b = a.pop_back() // b = 1; a = [2,3]
    a = {};          // a = []
  end
```

Методы очереди. **Все методы можно использовать в функциях**

```
<тип элементов> my_queue [$];

my_queue[$]: доступ к последнему элементу
my_queue[0]: доступ к нулевому элементу
my_queue[1]: доступ к первому элементу
my_queue = {}: удалить все элементы
my_queue.insert( <индекс>, <элемент>): вставить <элемент> на
место номер <индекс>
my_queue.delete( <индекс> ): удалить элемент на позиции
<индекс>
my_queue.pop_front(): достать нулевой элемент
my_queue.pop_back(): достать последний элемент
my_queue.push_front( <элемент> ): вставить начало <элемент>
my_queue.push_back( <элемент> ): вставить в конец <элемент>
my_queue.size(): количество элементов в очереди.
```

mailbox

mailbox -- класс для обмена данными между потоками.

```
mailbox #( int ) my_mb = new();
initial
  begin
    forever
      begin
        #5;
        my_mb.put( $urandom_range(10,1) );
      end
  end

int value;
initial
  begin
    forever
      begin
        #1;
        my_mb.get( value );
        $display( value );
      end
  end
```

Методы экземпляра **mailbox:**

mailbox #(**[тип данных]**) my_mb = new(**[размер]**);
(тип можно не указывать) (0/ничего = бесконечная очередь)

my_mb.put(): **ждать** пока появится место и поместить в очередь
my_mb.get(): **ждать** пока что-то появится и достать из очереди
my_mb.peek(): **ждать** пока что-то появится и скопировать первое сообщение из очереди (не доставать)

Можно использовать в функциях:

my_mb.try_put(): **попробовать** поместить в очередь. Не ждать.
my_mb.try_get(): **попробовать** достать из очередь. Не ждать.
my_mb.try_peek(): **попробовать** скопировать первое сообщение из очереди. Не ждать.

my_mb.num(): количество элементов в очереди.

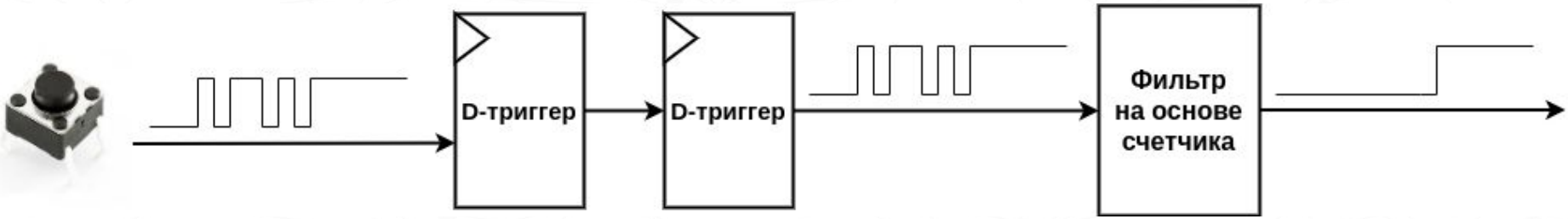
НТЦ Метротек

V. Разбираем фильтр дребезга (debouncer) и его testbench

НТЦ Метротек

Фильтр дребезга (debouncer)

Проект на github: https://github.com/stcmtk/fpga-webinar-2020/tree/master/lecture_3/debouncer



Не синхронный с тактовой частотой и **с помехами** сигнал.

Синхронизация с тактовой частотой. 2 триггера:
Первый может попасть в **метастабильное** состояние (не 0 и не 1) и это состояние может попасть дальше в схему.

Синхронный с тактовой частотой, но все еще с помехами.

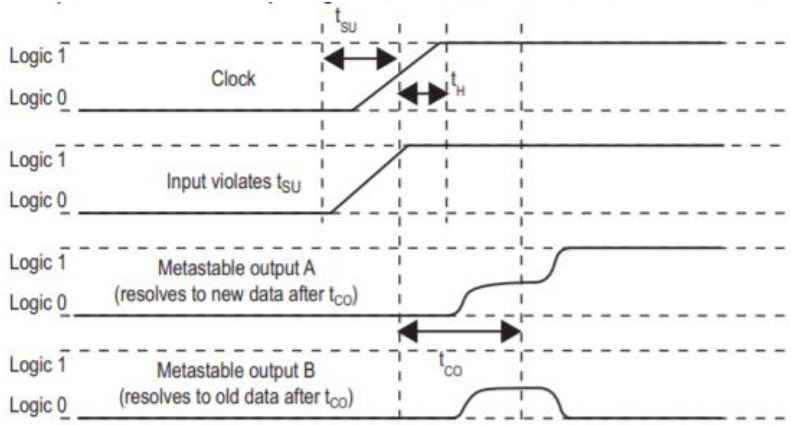
Сигнал должен не изменятся в течении заданного времени -- тогда считаем что значение стабильное.

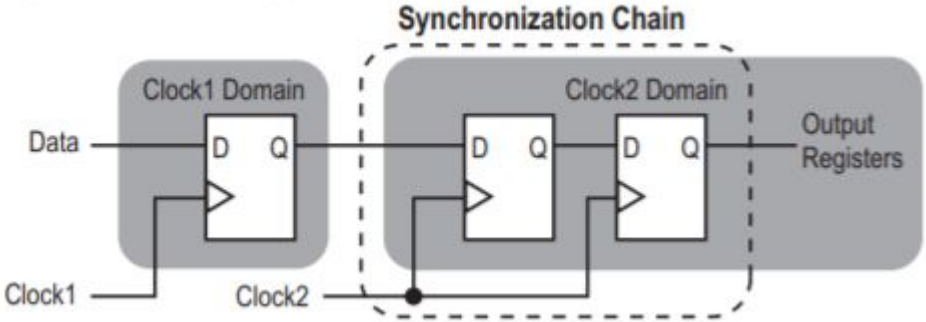
Синхронный с тактовой частотой и чистый сигнал.

НТЦ Метротек

Защелкивание асинхронных сигналов

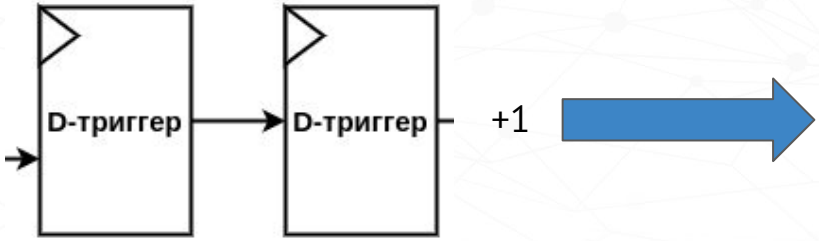
Understanding Metastability in FPGAs





НТЦ Метротек

Фильтр дребезга (debouncer)




```
logic [2:0] pin_d;

always_ff @( posedge clk_i )
  begin
    pin_d[0] <= pin_i;
    pin_d[1] <= pin_d[0];
    pin_d[2] <= pin_d[1];
  end
```

```
logic pin_differ;

// 0 -- когда пин не меняется за 2 такта
// 1 -- пин на двух тактах имеет разное значение
assign pin_differ = pin_d[2] ^ pin_d[1];
```

+

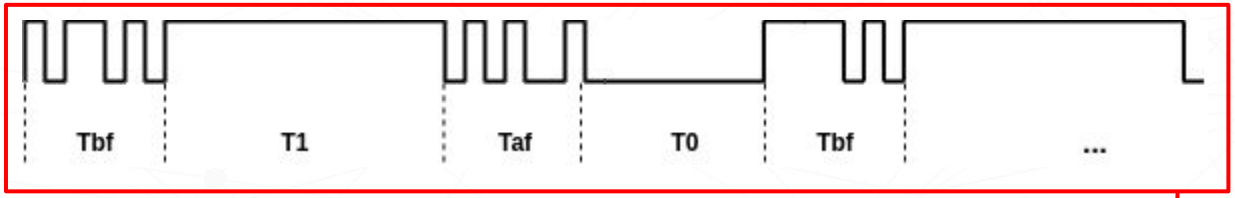
```
always_ff @( posedge clk_i )
  if( s_rst_i )
    db_counter <= '0;
  else
    begin
      if( pin_differ )
        db_counter <= '0;
      else
        db_counter <= db_counter + (DB_CNT_W)'(1);
    end
```

=

Фильтр на основе счетчика

НТЦ Метротек

Тестируем debouncer



Генерируем задания:
* Длительность перепадов до установки (**Tbf**)
* Длительность стабильного сигнала 1 (**T1**)
* Длительность после стабильного сигнала (**Taf**)
* Длительность стабильного сигнала 0 (**T0**)

Mailbox

Генерируем воздействия по заданиям

pin_i

clk_i

DUT
debouncer.sv

pin_state_o

Анализируем сигнал

Максимальная длительность "1"

Минимальная длительность "1"

Средняя длительность "1"